[0431] Code generators may be easily constructed from a well defined metamodel. Many CASE and UML tools already provide code generation based on metamodels. In those cases, a complex mapping is necessary to generate appropriate source code. The introduction of a meta-implementation layer simplifies code generation significantly. Operations cannot be fully captured in the current UML and CASE tools without resorting to a programmer typing code into the metamodel tool. A virtual implementation, on the other hand, contains a drag-and-drop interface for adding "pseudo-code" logic that the virtual implementation is able to execute directly. By constructing the virtual implementation for a model, a simple one-to-one mapping exists between the virtual implementation and the metamodel. By further constructing a serializer for each virtual implementation, source code can be generated for any language with similar concepts. A serializer is an object that writes the lines of code related to one virtual implementation. For example, a CSharp_ForLoopSerializer would accept a ForLoopImplementation and generate the appropriate C# source code as follows:

```
Write( "for( int i=" );
Serialize( ForLoopImplementation.getMinValue( ) );
Write( "; i<" );
Serialize( ForLoopImplementation.getMaxValue( ) );
Write( "; i=i+" );
Serialize( ForLoopImplementation.getIncrement( ) );
WriteLine( "{" );
//For-each sub-operation serialize it
    for( Iterator itr= ForLoopImplementation.getSubOperations( );
    itr.hasNext( ); ) {
        Serialize( itr.next( ) );
    }
    //close the loop
    Write( "}" );
    In the above code, each call to Serialize( ) gets the serializer
appropriate for the implementation object returned. The resulting code
would look something like this:
    for( int i=0; i<10; i++ ) {
      System.out.println( "Hello World" );
    }
```

[0432] Object query languages provide a simple scripting language, similar to SQL, for querying sets of objects. The query places constraints on the objects that should be returned and may also select attribute values from those objects.

[0433] Selection of various attributes of objects is simplified by using a component integration engine that describes every object as a model containing attributes. Retrieval of these attributes is performed using Attribute Accessors.

[0434] The value constraints and the logical expressions already developed in the meta-implementation layer form all the structures necessary for expressions used in the object selection constraints.

[0435] After implementing the language serializers according to the process described above the final component necessary to perform an object query language is a parser. Where the serializers convert virtual implementations to source code, parsers convert source code into a structure understood by the computer. In this case, a parser would need to be implemented to convert the Object Query Language (OQL) back into virtual implementations. Parsers are well understood by computer programmers. The reference component integration engine, component integration engine, even provides all the structures necessary to rapidly build a parser for a new language and convert that language into virtual implementations.

[0436] An object-relation mapping engine stores object data to records in a relational database. The process involves retrieving the object data, performing some simple conversions on this data to ensure that the data is in the proper form for storage, and generating the SQL statements necessary for inserting, updating, and deleting this data. Object-relational mapping engines also perform the mapping from the database back to an object. In this case, the object to be retrieved is selected by a process. This process must use the object's identity (the primary key in the database) or some of the object's characteristics (an OQL query). The records returned from the database are converted back to an object structure using the same mapping used to store them (conversions are reversed).

[0437] By using a component integration engine with a meta-implementation layer which uses databases to create instances and a second meta-implementation layer for objects (virtual or compiled), an object can be stored in a database by simply mapping from the attributes of one meta-implementation layer to the attributes of another meta-implementation layer. When the object meta-implementation is copied into the database meta-implementation, a record is created in the database and the attribute values are stored in each of the columns. More complex mappings involving more than one table would simply involve more than one database meta-implementation model or would use the facilities the database meta-implementation provides for mapping objects to database tables. All the implementation related to SQL generation, object selection, attribute retrieval, and conversion are already part of the component integration engine (with a database meta-implementation).

[0438] An Extract Transform Load (ETL) tool provides the ability to select data from a data source (a database table, comma delimited file, fixed-length file, spreadsheet) and perform transformations on that data (conversions, merges, value lookups, etc.) to create a new record to be loaded into a database. The entire process can then be saved and reused to add more data to that database table at some future date. The ETL tool may also allow scheduling of these extracts on a continuous basis. ETL tools are very complex. Metadata needs to exist to describe each transformation operation so it can be reconstructed. This data needs to be saved to a database or file in order to be persisted for future use. The transformations must describe an implementation of an operation that accepts parameters and returns a result. The transformation process may include branching statements and loops. The resulting data must be stored in the appropriate database table.

[0439] A component integration engine can contain components for each type of datasource to create an object representation of the data contained in that data source. As discussed in the Object-Relational Mapping section, data retrieval and data storage related to objects is a trivial extension of a component integration engine. By simply adding the virtual operation implementations from a component integration engine to an Object-Relational Mapping Engine, an ETL tool results.